Algorand Blockchain
Introduction to blockchain and smart contract design
Part I: The Algorand Consensus

Giuseppe Persiano

Università di Salerno
giuper@gmail.com

An overview

Objective

Design and implementation of a *Distributed Autonomous Organization (DAO)* on the blockchain.

Our DAO

- it holds a certain number of tokens `FOSAD22Token`
- anyone can buy the tokens at the current price
- governors can set the price
- governors can sell their right
- when all tokens are sold, DAO dissolves and shares the proceeds among its founders

Distributed: once setup, it runs on the distributed blockchain

Autonomous: no human supervision is needed once it is started

Centralized

One program running on one machine on the Internet

Trust assumption: One program is trusted

Decentralized

One program run by the whole network

Trust assumption: Two thirds of the network are trusted

How we leverage on widespread trustfulness

Algorand Consensus: the executive summary

Pure Proof of Stake (PPoS) Byzantine Agreement

- can tolerate malicious users
- no need for a central authority, as long as a super-majority of the stake is in non-malicious hands.
- extremely fast (currently, one new block every $\approx$ 4 sec)
- fork with probability $< 10^{-18}$
- immediate finality
- no delegation or binding of the stake
- requires minimal computational power per node
- carbon neutral

Source: https://developer.algorand.org/docs/get-details/algorand_consensus/

Digital currency

Double spending

Each digital coin can only be spent once.

- transaction based: transactions take coins as input and output coins
  need to keep track of which coins have not been spent yet
  coins input to a transaction are burned
  - Bitcoin keeps a list of *unspent transaction outputs* `UTXO`
- account based: transactions move coins from one account to another
  need to keep track of number of coins associated with each account

the concept of a *transaction* is basic in both models

Current implementations

- Physical coins and bills:
  - each individual is an account
  - an individual owns the coins in his pocket/wallet
  - a transaction moves money from one pocket to another
- Bank accounts:
  - a transaction moves money from one account to another
  - the bank keeps track of the balance in each account
  - **private, centralized and offline** ledger

We want **public, decentralized, on-line ledgers**

Consensus on who owns money

- Physical coins and bills:
  - Consensus is trivial: we all agree that what's in my pocket is mine
- Bank accounts:
  - Because the **BANK** says so

Decentralized OnLine Consensus

Decentralized OnLine Consensus

Consensus on transactions

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions
- Nodes of the network must reach consensus on the next block

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions
- Nodes of the network must reach consensus on the next block
- A new block is proposed by one node of the network

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions
- Nodes of the network must reach consensus on the next block
- A new block is proposed by one node of the network
- Which node?

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions
- Nodes of the network must reach consensus on the next block
- A new block is proposed by one node of the network
- Which node?
  - The selected node is the one that can provide the solution to a *public* puzzle

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions
- Nodes of the network must reach consensus on the next block
- A new block is proposed by one node of the network
- Which node?
  - The selected node is the one that can provide the solution to a *public* puzzle

Decentralized OnLine Consensus

Consensus on transactions

- A blockchain is a *public* ledger organized as a sequence of *blocks*
- A block is a set of transactions
- Nodes of the network must reach consensus on the next block
- A new block is proposed by one node of the network
- Which node?
  - The selected node is the one that can provide the solution to a *public* puzzle

Puzzles

- Efficiency: the cost of solving a puzzle or realizing that we cannot solve the puzzle
  - green vs non-green blockchains
- Multiple Solvers: the puzzle can be solved concurrently by more than one node or each puzzle can only be solved by one node.
  - block finality
- The block proposed by the selected node must be verified by the other nodes

Distributed vs Centralized

- Centralized consensus is easy
  - Banks have been successfully operating for centuries

- Decentralized consensus is hard
  - No trusted party
  - Only trust that a (super-)majority is honest
  - Majority of what??? Sybil attack
  - Bitcoin: computing power
  - Algorand: tokens
  - Cannot be faked and does not need identity management

Bitcoin vs Algorand

Bitcoin's Puzzle

Proof of Work:

- hard-to-invert hash function
- probability of selection proportional to computing power
- the same puzzle might be solved by two (or more) nodes

Algorand's Puzzle

Proof of Stake:

- each token *(algo)* throws a die with prob $1/N$ of winning
- winning token can be shown
- probability of winning proportional to number of tokens held
- probability of two algos throwing the winning die is about 10^{-18}

Proof of Work

Dwork and Naor, 1992
Hashcash by Adam Back, 1997

Combatting junk mail

- easy to compute *hash function* $H : \{0,1\}^\star \to \{0,1\}^\ell$
- infeasible to find a *collision*: $x \neq y$ such that

$$H(x) = H(y)$$

  and infeasible to invert
- add a `nonce` to an email message such that

$$H(\texttt{nonce}||\texttt{mail})$$

  starts with k zeros
- receiver looks for `nonce` in the message and if not found or incorrect, email is deemed junk

Rationale

- Sender must spend some time before sending out a message
- Receiver must spend (much less) time verifying a message
- Overhead acceptable for the regular email user
- Unfeasible if sending out thousands of email

Cryptographic Hash Functions

- Takes a string x of any length and returns a fixed length output $H(x)$
- Easy to compute (hundreds of megabytes per second)
- Infeasible to find a collision

$$H(x) = H(y)$$

  this implies that H is infeasible to invert
- For a given x, only way to find `nonce` such that

$$H(\texttt{nonce}||x)$$

  starts with k zeros is to try different values for `nonce` until one is found. Number of average tries is exponential in k.

Concrete examples: `RipeMD`, `SHA256`, `Keccak` (aka `SHA3`)

Proof of Work in Bitcoin

- Starting from
  - Hash of previous block
  - Hash of Transactions of Current Block
  - Timestamp

  Find `nonce` such that the hash value is smaller than `target`.
- Every 2016 blocks (about 2 weeks), the `target` is recomputed by setting `NewTarget` to

  ```
  OldTarget * MinToProduceLast2016Blocks/20160Min
  ```

  to keep average time between blocks to around 10 minutes

Proof of Work in Bitcoin

- Starting from
  - Hash of previous block
  - Hash of Transactions of Current Block
  - Timestamp

  Find `nonce` such that the hash value is smaller than `target`.
- Every 2016 blocks (about 2 weeks), the `target` is recomputed by setting `NewTarget` to

  `OldTarget * MinToProduceLast2016Blocks/20160Min`

  to keep average time between blocks to around 10 minutes

`0x0000000000000000000901BA0000000000000000000000000000000000000`

Economic Incentives

A native currency is associated with the Bitcoin blockchain to reward nodes that produce (mine) new blocks

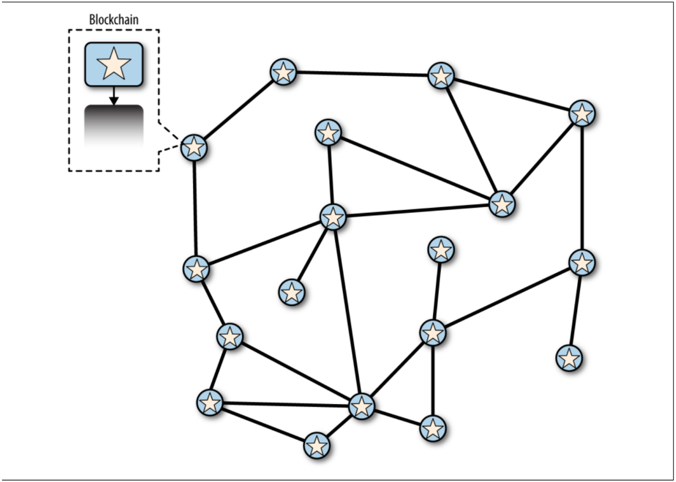
- miner receives the transaction fees associated to each transaction in the block
- a certain number of freshly mint bitcoins
  - initially 50 bitcoins per block
  - halving every 210_000 blocks
  - currently 6.25 bitcoins
  - coinbase transaction, only type of Bitcoin transaction with no input

Extending the blockchain

- At any given time, each node has its own view of the blockchain
- Need not to be one chain
- More chains are possible
- Where to attach the next block?
  - Nakamoto consensus: longest chain available

Forks

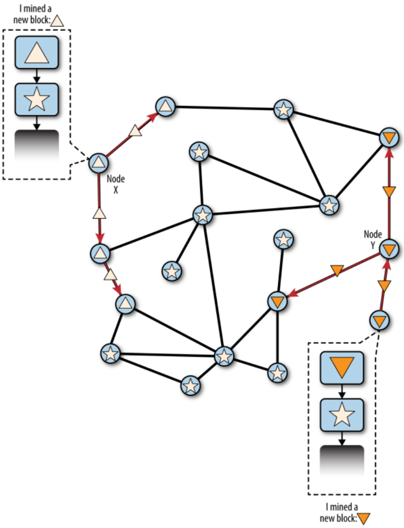
Suppose all nodes have the same view of the block chain



Pictures from *Mastering Bitcoin* by Andreas M. Antonopoulos.

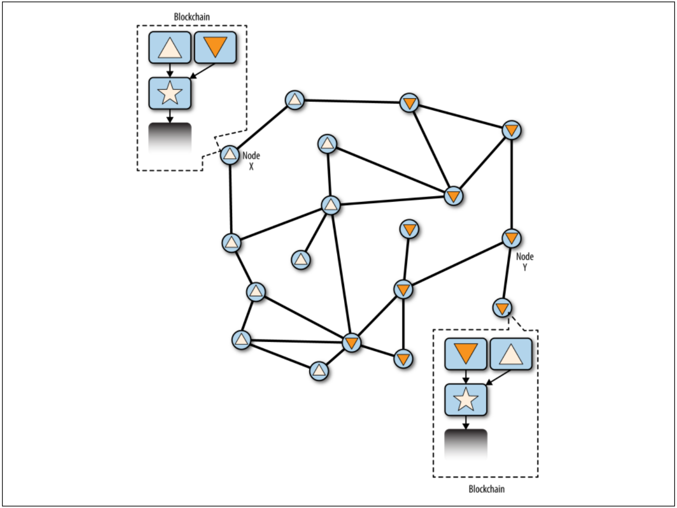
Forks

Two blocks found simultaneously



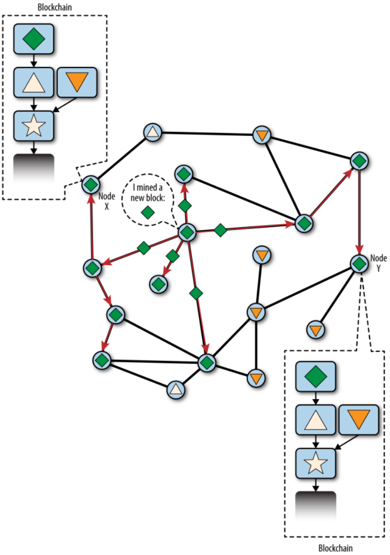
Forks

Two different views of the blockchain coexist



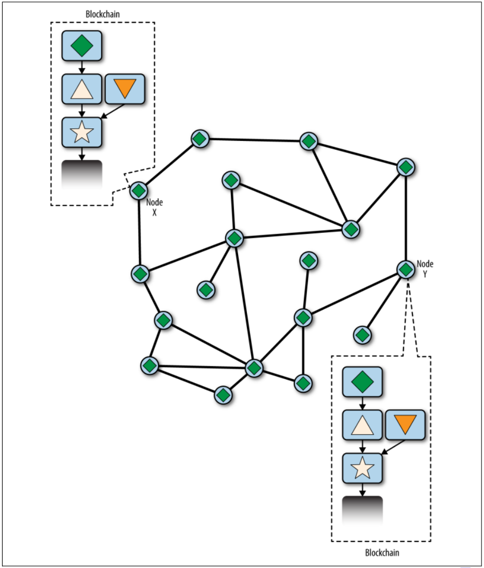
Forks

A new block extends one of fork



Forks

The network converges to one fork and the other is forgotten



Finality

- the transactions in the upside-triangle block disappear from the blockchain and must be resubmitted (unless they are also in the triangle block)
- theoretically the fork can extend to two blocks, if two blocks are found almost simultaneously by nodes on opposite side of the fork
- One-block forks occur about every day
- Two-block forks occur about every few weeks
- It is safe to wait 3 blocks before considering a transaction final
  - transaction finalization is about 30 min

The Algorand Consensus

Byzantine Agreement — Pease, Shostak and Lamport, 1980

A protocol to reach agreement

Works if at least 2/3 of the players are honest

- Honey Badger, Miller et al., CCS 2016
  - A designated set of servers reaches consensus through BA
  - A single point of attack
- Bitcoin-NG, Eyal et al., NSDI 2016
  - Use Nakamoto consensus to elect a leader
  - Leader publishes transactions
- ByzCoin, Kokoris-Kogias, Usenix Security Symposium, 2016
  - Use Nakamoto consensus to elect a group of participants
- Stellar, D. Mazieres, 2016
  - Each user trusts a subset of other users
  - Consistent agreement is reached if the transitive closure of the trust relations covers quorum of the nodes

BA$\star$

Cryptographic primitives

- Digital Signatures
  - used to sign messages
- Verifiable Random Functions
  - used to select users

Digital Signatures

- Digital equivalent of a signature

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user
  - Everybody can verify the signature

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user
  - Everybody can verify the signature
- To setup the digital signature, a user executes

$$(\mathtt{sigk}, \mathtt{vk}) \leftarrow \mathtt{GenKey}()$$

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user
  - Everybody can verify the signature
- To setup the digital signature, a user executes

$$(\mathtt{sigk}, \mathtt{vk}) \leftarrow \mathtt{GenKey}()$$

  - a signing key $\mathtt{sigk}$ and a verification key $\mathtt{vk}$

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user
  - Everybody can verify the signature
- To setup the digital signature, a user executes

$$(\mathtt{sigk}, \mathtt{vk}) \leftarrow \mathtt{GenKey}()$$

  - a signing key $\mathtt{sigk}$ and a verification key $\mathtt{vk}$
  - the verification key is made public (and linked to the identity of the user, if need be)

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user
  - Everybody can verify the signature
- To setup the digital signature, a user executes

$$(\mathtt{sigk}, \mathtt{vk}) \leftarrow \mathtt{GenKey}()$$

  - a signing key $\mathtt{sigk}$ and a verification key $\mathtt{vk}$
  - the verification key is made public (and linked to the identity of the user, if need be)
- to sign message m, user executes

$$\mathtt{sig} = \mathtt{Sign}(m, \mathtt{sigk})$$

Digital Signatures

- Digital equivalent of a signature
  - The legitimate user can sign a document
  - Infeasible for others to sign on behalf of the legitimate user
  - Everybody can verify the signature
- To setup the digital signature, a user executes

$$(\mathtt{sigk}, \mathtt{vk}) \leftarrow \mathtt{GenKey}()$$

  - a signing key $\mathtt{sigk}$ and a verification key $\mathtt{vk}$
  - the verification key is made public (and linked to the identity of the user, if need be)
- to sign message m, user executes

$$\mathtt{sig} = \mathtt{Sign}(m, \mathtt{sigk})$$

- given m and $\mathtt{sig}$ anybody can check that $\mathtt{sig}$ is a signature of m w.r.t. to $\mathtt{sigk}$ by executing

$$\mathtt{Verify}(\mathtt{vk}, \mathtt{sig}, m)$$

Digital Signatures

Digital Signatures in Algorand

- an *address/user* is a *verification key*
- a user *owns* an address, if they know the associated *signing key*
- each Algorand address is associated with a certain number of Algo tokens: the balance
- if a user proposes a transaction, i.e. to *transfer/spend* some tokens, the transaction is signed with the *signing key* associated with the address
- everybody can verify that a transaction is properly signed

signing key aka the *spending key*

Algorand uses Ed25519: EdDSA with SHA-512 and Curve 25519.

Verifiable Random Functions

[Micali, Rabin, and Vadhan, 1999]

- A Cryptographic Primitive:
  - a public key $\mathtt{pk}$
  - a secret key $\mathtt{sk}$
- Given value x and $(\mathtt{pk}, \mathtt{sk})$
  - it is possible to compute $y = F(\mathtt{sk}, x)$ and a proof Π that y is the correct value
  - for all $x \neq x'$, $y = F(\mathtt{sk}, x')$ is still *random* even conditioned on $\mathtt{pk}, (x, y)$
- Given (x, y, Π) and a public key $\mathtt{pk}$
  - it is possible to verify that y is the correct value

VRF used by Algorand ⇒ https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vrf-10

The Algorand Consensus

Rationale

- Addresses (i.e., verification keys) are the players
- Each player has a weight proportional to its stake (number of Algo tokens associated with the address)
  - Creating dummy addresses is useless
- Digital Signatures
  - used to sign messages exchanged during the consensus protocol
- Verifiable Random Functions
  - used to select players with probability proportional to its stake

Cryptographic Sortition and Participant Replacement

- Cryptographic Sortition
  - Each party evaluates the VRF on some public information of the blockchain to check if selected for a committee
    - if not selected, no action
    - if selected, announce and every one can verify correctness
- Participant Replacement
  - once selected, a participant sends one message
  - no time for an adversary to attack a committee member

Cryptographic Sortition

- τ (expected) number of users to be chosen
- user i with $(\mathtt{pk}_i, \mathtt{sk}_i)$ is selected with probability proportionally to the weight w_i
  - more than one sub-user $(i, 1), \ldots, (i, w_i)$ can be chosen
- for role `role`

Algorithm for Cryptographic Sortition

let s be the seed and W total weight

- set $(h, \Pi) = F(\mathtt{sk}_i, s||\mathtt{role})$ and $p = \tau/W$
- return (h, Π, j) such that $h/2^\ell$ falls in interval

$$\left[\sum_{k=0}^{j-1} B(k; w_j, p), \quad \sum_{k=0}^{j} B(k; w_j, p)\right]$$

where $B(k; w, p) = \binom{w}{k} p^k (1-p)^{w-k}$ is the probability that k out of w_i are selected.

Sybil attacks

Splitting the weight w of a user into $w = w_1 + w_2$ of two dummy users does not help

$$B(k_1; w_1, p) + B(k_2; w_2, p) = B(k_1 + k_2; w_1 + w_2, p)$$

Selecting the seed

- at round $r-1$, every block proposer u adds also the proposed seed s_r for round r computed as

$$(s, \Pi) = F(\mathtt{sk}_u, s_{r-1-(r \bmod R)} || r)$$

- s_0 chosen at start
- key $\mathtt{sk}_u$ must be chosen in advance
  - weak synchrony assumption

Participation Keys

- To take part in consensus, a user must be online
- This means that some secret key of the user should be given to a node
- Users should not use the *spending key* for consensus.
  If the node is compromised, the user could lose the Algo tokens
- Instead...

Participation Keys

- a user generates and registers a *participation key* for a certain number of rounds
- it also generates a collection of *ephemeral* keys, one for each round, signs these keys with the participation key, and then deletes the participation key.
- each ephemeral key is used to sign consensus messages for the corresponding round, and is deleted after the round is over.
  - Using participation keys ensures that a user's algos are secure even if their participating node is compromised.
  - Deleting the participation and ephemeral keys after they are used ensures that the blockchain is forward-secure and cannot be compromised by attacks on old blocks using old keys.

Consensus

1. Block Proposal
2. Soft Vote
3. Certify Vote.

All messages in each phase are signed with one of the ephemeral keys of the round.

I - Block proposal

- A user runs with cryptographic sortition with $\tau = 26$.
  - with probability $1 - 10^{-11}$ we have at least one proposer and at most 70 proposers
- Each user propagates block k from $1, \ldots, j$ with highest priority

$$H(VRF||k)$$

along with proof and proposed block

I - Block Proposal

- Each node get proposals from other nodes
- Verify the signature of the block and the cryptographic sortition
- Do not propagate blocks with lower priority than already seen
- This is done for a fixed amount of time
  - tradeoff between efficiency and probability of not receiving any proposed block

II - Soft Vote

- Each node runs cryptographic sortition with `role=‘‘committee||round||step’’` to see if chosen for the soft vote committee
- All chosen accounts will have a weighted vote based on the number of algos the account has
- These votes will be for the highest priority block proposed and will be sent out to the other nodes signed and along with the sortition proof
- If within timeout no quorum is formed
  - A new committee is formed until a quorum is reached
  - for a maximum number of steps

References

- Y. Gilad et al., *Algorand: Scaling Byzantine Agreement for Cryptocurrencies*, SOSP '17
- J. Chen, S. Micali, *Algorand: A secure and efficient distributed ledger*, TCS, 2019
- Algorand Blockchain Feature Specification, 2019
- Algorand Source Code at `github/algorand/go-algorand`

PoW Consensus vs PoS Consensus

PoW vs PoS

- Computational Power
  - PoW: very expensive (inverting hash function by brute force)
  - PoW: need to be compensated
  - PoS: inexpensive (can run on Raspeberry pi)
  - PoS: no need to be compensated
  - PoS: more tokens, more likely to be selected, more work
  - PoS: more tokens, greater interest in the network, more willing to be involved
- Robustness to attacks
  - PoW and PoS rely on majority of good players
  - Gain majority to disrupt the networks
  - PoW: Buy hardware
  - PoS: Buy tokens
    - if you own more than half of the PoS economy
      why would you want to disrupt it?